The Composite Component-Based Operating System

Gabriel Parmer

Based on research with:
Runyu Pan, Yuxin Ren, Phani Kishore Gadepalli,
Wenyuan Shao, Qi Wang, Jiguo Song, and many others

The George Washington University
gparmer@gwu.edu
Requirements Convergence

Embedded Systems
Predictability
Simplicity
SWaP-C

Multi-tenant Cloud
Performance
Isolation
Elasticity
Requirements Convergence

- Embedded Systems
  - Predictability
  - Simplicity
  - SWaP-C
- Predictability/Tail-latency
- Performance
- Isolation & Security
- Multi-tenant Cloud
  - Performance
  - Isolation
  - Elasticity
Requirements Convergence

- Data-centers
- Cyber-physical Systems
- Edge Computing
- Embedded Systems
- Performance
- Predictability
- Isolation
Component-based System Design

- Code, data, user-level
- Export APIs ($E_1 = \{ fn, \ldots \}$)
- Explicit dependencies
- Unit of reuse & isolation

- Minimize functionality for the necessary APIs
- Strong, fine-grained isolation
Component-based System Design

- System *composed* from components

- Limit *scope* of
  - compromises
  - faults
  - unpredictability
Component-based System Design

- System *composed* from components
- Limit *scope* of
  - compromises
  - faults
  - unpredictability
...a concept is *tolerated inside the μ-kernel* only if *moving it outside the kernel*, i.e. permitting competing implementations, would *prevent the implementation* of the system’s required functionality.”

- Liedtke '95
Guidance for μ-Kernel Design

"...a concept is tolerated inside the μ-kernel only if moving it outside the kernel, i.e. permitting competing implementations, would prevent the implementation of the system’s required functionality."

- Liedtke '95

**Composite**: push this to the extreme w/ component-defined
- scheduling
- parallel scalability
- concurrency
- capability delegation/revocation
- ...

...
Composite Kernel Objects

Capability-table nodes

Page-table nodes
Composite Kernel Objects

- Capability-table nodes
- Page-table nodes
- Components
Composite Kernel Objects

- Capability-table nodes
- Page-table nodes
- Components
- Threads
Composite Kernel Objects

- Capability-table nodes
- Page-table nodes
- Components
- Threads
- Untyped Memory
Composite Kernel Objects

- **Capability-table nodes**
- **Page-table nodes**
- **Components**
- **Threads**
- **Untyped Memory**
Composite Kernel Objects

- Capability-table nodes
- Page-table nodes
- Components
- Threads
- Untyped Memory
- Virtual Memory
Composite Kernel Objects

- Capability-table nodes
- Page-table nodes
- Components
- Threads
- Untyped Memory
- Virtual Memory
- IPC: sync & async
Asynchronous IPC

End-to-end timing requires dependency analysis
Synchronous IPC between Threads

End-to-end timing based on kernel mechanisms
- In-kernel priority inheritance machinery, ceiling w/ limited prio
- Budget management and expiration
Spectrum of IPC Mechanisms

- Liedtke's L4
- seL4 + MCS
- Nova
- Fiasco L4
- Composite

Synchronous Rendezvous Between Threads

Thread Migration

Ford et al. *Evolving Mach 3.0 to a Migrating Thread Model*, USENIX Winter ’94
Thread Migration

$p_c$
Thread Migration

$p_c$
Thread Migration

Thread Migration

scheduler

$p_c$
Thread Migration

scheduler

$p_c$
Thread Migration
Thread Migration

Parmer et al. *Predictable Interrupt Management and Scheduling in the Composite Component-based System*, RTSS '08
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17
Thread Migration

Thread Migration

Parmer et al. *Predictable Interrupt Management and Scheduling in the Composite Component-based System*, RTSS '08
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17
Thread Migration

Parmer et al. *Predictable Interrupt Management and Scheduling in the Composite Component-based System*, RTSS '08
Gadepalli et al. *Temporal Capabilities: Access Control for Time*, RTSS '17
Thread Migration

Gadepalli et al. Temporal Capabilities: Access Control for Time, RTSS '17
Composite Kernel Objects

Threads

temporal capabilities

IPC: **sync** & **async**

registrations

schedulers

b, p
Thread Migration

ULS too slow?

Gadepalli et al. Slite: OS Support for Near Zero-Cost, Configurable Scheduling, RTAS '20
Thread Migration

ULS too slow?

Dispatcher latency: 41 cycles

Gadepalli et al. *Slite: OS Support for Near Zero-Cost, Configurable Scheduling*, RTAS ’20
Thread Migration

IPC too slow?
Thread Migration

Round-trip IPC | seL4 | composite
--- | --- | ---
X86-32 (3.2GHz) | 934 | 741
Cortex a9 (667 GHz, zynq) | 630 | 543

Gadepalli et al. *Chaos: a System for Criticality-Aware, Multi-core Coordination*, RTAS ’19
Kernel Synchronization

Can kernel APIs/implementation
- Limit the scalability of components?
- Cause interference between components?
Kernel Synchronization

Can kernel APIs/implementation
- Limit the scalability of components?
- Cause interference between components?
Kernel Synchronization

Can kernel APIs/implementation
- Limit the scalability of components?
- Cause interference between components?
Kernel Synchronization

Component policies for predictability/scalability?

- Kernel must be **wait-free** w/ component-controlled cacheline contention
- Component-controlled IPI facilities

Gadepalli et al. *Chaos: a System for Criticality-Aware, Multi-core Coordination*, RTAS ’19
Wang et al. *Speck: A Kernel for Scalable Predictability*, RTAS ’15
Research Approaches
Research Approaches

**Composite Infrastructure**
- PoLP-focused RTOS
- NetBSD Rumpkernels
- Xen-like driver domains
- NASA CFS
- Mixed-criticality system orchestration
Research Approaches
ucVM Challenges

- MPU-based isolation vs. page-table software abstractions
  - path-compressed radix tries flattened into MPU regions
- Limited # of MPU protected regions
  - Solve memory layout: contiguous memory fits into # regions
  - Treated as dynamic software protection cache
- Efficient coordination and event processing
  - Kernel-bypass + micro-optimization
- ~8 VMs in 128KiB SRAM
Also: Secure Bare-metal interrupts

- Use Trustzone-M as secure kernel bypass for interrupts

Pan et al. *Predictable Virtualization on Memory Protection Unit-based Microcontrollers*, RTAS ’18

Research Approaches

A_1  A_2
OS

A_1  A_2
RTOS

Density
Multi-tenancy
Latency
## The Edge: Options

<table>
<thead>
<tr>
<th></th>
<th>Isolation</th>
<th>Scalability</th>
<th>Startup Time</th>
<th>High-performance networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes</td>
<td>✗</td>
<td>✓</td>
<td>!</td>
<td>!</td>
</tr>
<tr>
<td>Containers</td>
<td>!</td>
<td>!</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>VMs</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
## The Edge: Options

<table>
<thead>
<tr>
<th></th>
<th>Isolation</th>
<th>Scalability</th>
<th>Startup Time</th>
<th>High-performance networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Containers</td>
<td>!</td>
<td>!</td>
<td>✗</td>
<td>!</td>
</tr>
<tr>
<td>VMs</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>EdgeOS: Feather-weight Processes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
**The Edge: Options**

<table>
<thead>
<tr>
<th>Isolation</th>
<th>Scalability</th>
<th>Startup Time</th>
<th>High-performance networking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EdgeOS:</strong> Isolation, Predictability, Performance, and Scale</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• High speed data movement (10Gbps+) without sacrificing isolation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Startup &gt; <strong>100X</strong> faster than fork+exec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Scales to 1000s of services per host = 1 service per user!</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| EdgeOS: Feather-weight Processes | ✓ | ✓ | ✓ | ✓ |
EdgeOS: Strong Isolation & Performance
Per-client Service Instantiation

- Docker: the execution time of “docker start”
- Firecracker: the start time of the recommended “hello” image
- Linux: fork() + exec()
Per-client Service Instantiation

- Docker: the execution time of “docker start”
- Firecracker: the start time of the recommended “hello” image
- Linux: fork() + exec()
Per-client Service Instantiation

- Docker: the execution time of “docker start”
- Firecracker: the start time of the recommended “hello” image
- Linux: fork() + exec()
EdgeOS: scalability for TLS proxies

![Graph showing throughput for EOS and Linux](image)
Conclusions

Data-centers

Cyber-physical Systems

Edge Computing

Performance

Isolation

Predictability

A1

A2

A1

A2

os

RTOS

A1

A2
https://github.com/gwsystems/composite
https://composite.seas.gwu.edu